

BUNDESREPUBLIK DEUTSCHLAND



JG996 U 8 pro
09/029299
04/09/01

Prioritätsbescheinigung über die Einreichung einer Patentanmeldung

Aktenzeichen: 198 46 673.0

Anmeldetag: 9. Oktober 1998

Anmelder/Inhaber: Infineon Technologies AG, München/DE
Erstanmelder: Siemens AG, München/DE

Bezeichnung: Verfahren zur Verbindung von Stackmanipulations-
angriffen bei Funktionsaufrufen

IPC: G 06 F 12/14

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der
ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 20. März 2001
Deutsches Patent- und Markenamt
Der Präsident
im Auftrag

[Signature] CERTIFIED COPY OF
PRIORITY DOCUMENT

Beschreibung

Verfahren zur Verbindung von Stackmanipulationsangriffen bei Funktionsaufrufen

5

Auf zukünftigen Chipkarten sollen voneinander abhängige Softwaremodule verschiedener Hersteller installiert werden. Unterschiedliche Module unterschiedlicher Hersteller haben dabei unterschiedliche Zugriffsrechte auf Ressourcen der Chipkarte. Beispielsweise hat nur das Betriebssystem Zugriff auf bestimmte Speicherbereiche im NVRAM, die von anderen Modulen nicht manipuliert werden dürfen.

10 Eine solche Zugriffsbeschränkung zum Schutz der Arbeitsspeicherbereiche einzelner Programme vor veränderndem Zugriff

15 durch andere auf der Chipkarte ablaufende Programme, ist beispielweise in dem U.S.-Patent 5,754,726 beschrieben. Das dort beschriebene Verfahren stellt zwar sicher, daß von einem auf der Chipkarte aktiven Programm keine Arbeitsspeicherbereiche anderer Programme manipuliert werden können. Eine weitere Angriffsmöglichkeit besteht jedoch darin, nicht den Arbeitsspeicher, sondern den Stapspeicher mit den jeweiligen Rücksprungadressen der Unterprogramme zu verändern. Ein solcher Manipulationsangriff auf den Stapspeicher (Stack) des Prozessors, kann mit dem Verfahren der U.S.-PS 5,754,762 nicht abgewehrt werden.

20 Aus Speichereffizienz- und Performancegründen liegen die Stacks von aufgerufener und aufrufender Funktion nämlich physikalisch im selben Speicherbereich hintereinander. Da konzeptionell nicht ausgeschlossen werden kann, daß eine Bibliotheksfunktion auf hohem Sicherheitsniveau eine Funktion einer

25 Applikation mit niedrigem Sicherheitsniveau aufruft, besteht ein mögliches Angriffsszenario darin, daß die aufgerufene Funktion der Applikation durch Zugriff auf den Programmstack den Datenbereich der Bibliotheksfunktion auf dem Stack manipuliert.

35

Auf Chipcard Controllern ist eine Lösung im Stand der Technik bisher nicht vorhanden. Die Problemstellung ist neu, da bisher ein Hersteller für die gesamte Software zuständig war.

5

In modernen Prozessoren wird z.B. eine Page Table oder Segment Descriptor Table (MMU) verwendet, in die das Multitasking-Betriebssystem den für die Applikation gültigen Speicherbereich einträgt. Die Prozeßkommunikation und -überwachung wird dabei vom Betriebssystem durchgeführt.

10

Funktionsaufrufe zwischen Funktionen auf unterschiedlichen Sicherheitsniveaus gehen bei Chipkarten aus Performancegründen nicht über das Betriebssystem, sondern werden direkt ausgeführt.

15

Es ist daher Aufgabe der Erfindung, die direkte und indirekte Manipulation des Stack-Speicherbereichs als sicher bewerteter Funktionen durch als unsicher bewertete Funktionen zu verhindern.

20

Erfindungsgemäß wird diese Aufgabe durch ein Verfahren gelöst, bei dem der Stackzugriff bei einem Aufruf einer unsicheren Funktion durch Hardware auf deren Stackbereich beschränkt wird.

25

Es ist dabei besonders bevorzugt, zur Beschränkung des Stackzugriffs vor dem Aufruf der unsicheren Funktion die Referenz auf den Stackframe der aufrufenden Funktion zu speichern.

30

Weiter ist es dabei bevorzugt, einen Mechanismus vorzusehen, durch den verhindert wird, daß die aufgerufene Funktion den Wert der Referenz auf den Stackframe verändern kann. Weiter ist es bevorzugt, durch einen Schutzmechanismus sicherzustellen, daß der Stackpointer nicht den gültigen Stackbereich der aktuellen Funktion überschreitet.

35

Besonders bevorzugt ist es, bei dem Rücksprung aus der unsicheren Funktion den ursprünglichen Zustand auf den Stack wiederherzustellen.

5 Das erfindungsgemäß günstigste Verfahren läuft so ab, daß beim Funktionsaufruf zunächst auf dem Stack ein Speicherbereich für zu schützende Funktionsdaten reserviert und optional die Funktionsargumente dahinter auf den Stack gelegt werden und die im geschützten Bereich liegende Referenz auf den
10 Stackframe der aufrufenden Funktion auf den zuvor reservierten Bereich des Stacks gelegt und die Referenz auf den Stackframe der aufgerufenen Funktion in den geschützten Bereich geschrieben wird.

15 Im folgenden wird die Erfindung anhand des in den beigefügten Zeichnungen dargestellten Ausführungsbeispiels näher erläutert. Es zeigen:

Fig. 1 die Belegung des Stacks und der zugehörigen Register
20 vor und nach dem Funktionsaufruf; und

Fig. 2 schematisch den Ablauf der Aufrufe bei einem abgesicherten Funktionsaufruf.

25 Bisher lieferte ein einziger Chipkartenhersteller sowohl das Betriebssystem der Chipkarte als auch die Anwendungsprogramme. Das Betriebssystem der Chipkarte konnte also als ein Teil des Anwendungsprogramms gesehen werden. Das Chipkartenbetriebssystem und die Anwendungsprogramme wurden auf speziell
30 hergestellten Masken für das ROM (Nurlesespeicher) der Chipkarten-IC's geliefert. Es handelte sich also bisher um Lösungen, bei denen die Programme im wesentlichen hardwaremäßig, also fest verdrahtet, vorgegeben waren.

35 Die vorliegende Erfindung behandelt demgegenüber eine Situation, bei der verschiedene Hersteller Bibliotheken und Anwendungsprogramme liefern können, die dann auf einer Karte ko-

existieren müssen. Aus Sicherheitsgründen hat die Programmarchitektur der Chipkarte dann zu ermöglichen, das Betriebssystem und die Bibliotheken gegen Manipulationen von deren eigenen "privaten" Daten, Programmcode und Stack durch ein laufendes Programm zu schützen. Dies kann bei einem Ausführungsbeispiel der Erfindung durch verschiedene Maßnahmen erreicht werden:

1. Segmentierte Adressierung:

10

Der physikalische Adreßraum von 2^{24} Bit wird über maximal 256 Daten- und 255 Programmsegmente zugänglich gemacht. Die Länge des physikalischen Adreßraums, der von einem Segment adressiert werden kann, kann zwischen 4 Byte und 2^{16} Byte liegen.

15

Ein Segment wird durch seine Länge und seine physikalische Anfangsadresse definiert. Die Adresse (Pointer) eines Speicherplatzes besteht dann aus 8 Bit, die die Adresse des Segments darstellen und einem Offset aus 16 Bit. Dies bildet die direkte Adresse.

20

2. Speicherverwaltungseinheit (Memory Management Unit = MMU):

25

Eine Speicherverwaltungseinheit (MMU) unterhält eine Liste aller Segmente, die von den laufenden Programmen benötigt werden. Jedes dieser Segmente verfügt über zusätzliche Attribute in der MMU, wie seine Eigenschaft als "Programmsegment" oder "Datensegment", Identifizierung des Programms, zu dem es gehört, und Vertrauensklasse. Verschiedene Programme, die zur gleichen Zeit laufen, werden durch unterschiedliche Programmidentifizierungen unterschieden. Programmzähler und Datenadressen beziehen sich immer auf Segmenteinträge in der MMU mit der gleichen Programmidentifizierung. Weiterhin verweist das Segment des Programmzählers auf einen Eintrag in der MMU mit der gleichen Segmentidentifikation und dem Attribut "Programmsegment". Andererseits können alle Datenadressen in der MMU-Eintragungsliste über die gleiche Programmidentifizierung und das Attribut "Datensegment" gefunden werden.

3. Vertrauensklassen:

Wenn ein Hersteller ein Programm A schreibt, das auf ein anderes Programm B eines anderen Herstellers zugreift, dann vertraut der erste Hersteller automatisch dem Code des zweiten Herstellers. Andernfalls würde er nicht auf dieses Programm zugegriffen haben. Andererseits wissen die Hersteller des Programms B nicht notwendigerweise irgend etwas über das Programm A. Daher müssen sie ihren Programmcode, den Stackinhalt und die Daten gegen schädliche Manipulationen durch Programm A schützen. Dies muß insbesondere deshalb garantiert werden, weil die Bibliothek B auch von anderen Anwendungsprogrammen benutzt werden kann, die sich auf eine korrekte Funktion der Bibliothek B verlassen. Erfindungsgemäß wird dieser Schutz durch vier Vertrauensklassen (0 bis 3) die zusätzliche Attribute der Segmenteinträge in der MMU darstellen, unterstützt. Ein Programmabschnitt auf einer niedrigeren Vertrauensklasse genießt größeres Vertrauen als ein anderer Programmabschnitt mit einer höheren Vertrauensklasse. So können Gerätetreiber auf einem Segment mit der Vertrauensklasse 0 liegen, das Kartenbetriebssystem auf Segmenten mit einer Vertrauensklasse 1, Bibliotheken auf der Vertrauensklasse 2 und Anwendungen auf der Vertrauensklasse 3. Die Vertrauensklassen spielen eine wichtige Rolle beim Aufstellen von Regeln für Funktionsaufrufe zwischen Segmenten (ferne Aufrufe) und dem Datenzugriff.

4. Funktionseintrittstore:

Nur Funktionen, die von dem selben Hersteller einer Bibliothek oder Anwendung hergestellt worden sind, werden in ein Segment gepackt. Deshalb braucht ein Segment keine Schutzmaßnahmen für Funktionsaufrufe innerhalb des Segments (nahe Aufrufe) vorzusehen. Andererseits sind ferne Aufrufe potentiell gefährlich. Einem Anwendungsprogramm darf es nicht erlaubt werden, einen Programmcodeabschnitt des Kartenbetriebssystems

an einer beliebigen Eintrittsadresse anzuspringen. Wenn dies nämlich nicht verhindert wird, können unvorhersehbare Vorgänge ablaufen. Die derzeit bevorzugte Lösung besteht darin, Adressen für ferne Aufrufe nicht als Funktionseintrittsadressen, sondern als Funktionstore zu definieren. Ein Segment kann maximal 255 solcher Tore besitzen. Wenn ein Fernaufruf erfolgt, besteht die Toradresse aus einem Wort von zwei Byte Länge: Das höherwertige Byte enthält die Segmentidentifizierung und das niedrigere Byte das Tor der Funktion, das angesprungen werden soll. Der Fernaufruf liest das Wort automatisch mit dem unter 2. definierten Offset des entsprechenden Segments und interpretiert es als Funktionseintrittsadresse. Trotzdem ist die Rückkehr von einem fernen Funktionsaufruf möglicherweise gefährlich, weil die Rückkehradresse eine direkte Adresse auf dem Stapspeicher (Stack) darstellt, die von der aufgerufenen Funktion verändert worden sein kann. Deshalb werden üblicherweise nur Funktionsaufrufe von höheren Vertrauensklassen auf niedrigere Vertrauensklassen erlaubt. Umgekehrt sind nur Fernrücksprünge von niedrigeren Vertrauensklassen auf höhere Vertrauensklassen zulässig. Eine Ausnahme sind die Funktionsaufrufe von den Vertrauensklassen 0 oder 1, die im folgenden diskutiert werden.

Obwohl es sich bei den Funktionsaufrufen üblicherweise um Funktionsaufrufe innerhalb des Segments oder Funktionsaufrufe auf derselben oder einer niedrigeren Vertrauensklasse handeln wird, wäre es zu restriktiv, Funktionsaufrufe von höheren auf niedrigere Vertrauensklassen vollständig zu verbieten: Das Kartenbetriebssystem muß in der Lage sein, ein Anwendungsprogramm zu starten. Auch das Protokoll zum Laden von Anwendungsprogrammen kann Rückrufe benötigen, bei denen ein Funktionszeiger einer Funktion A auf einer höheren Vertrauensklasse an eine Funktion B auf einer niedrigeren Vertrauensklasse übergeben wird und Funktion B nachher Funktion A aufruft. Aufrufe von niedrigeren auf höhere Vertrauensklassen werden im folgenden kurz als "Rückrufe" bezeichnet. Ebenso benötigt die virtuelle Javamaschine (JVM) solche Rückrufe, um

ein JavaKärtchen zu starten. Grundsätzlich erlauben wir jeden fernen Aufruf, verbieten aber ferne Rücksprünge von einer höheren Vertrauensklasse in eine niedrigere Vertrauensklasse.

Da auch Fernaufrufe von dem Kartenbetriebssystem auf ein Anwendungsprogramm oder eine Bibliothek inhärent unsicher sind, muß das Kartenbetriebssystem einen speziellen Mechanismus vorsehen, um sichere Rückrufe auszuführen. Dazu wird eine Kartenbetriebssystemfunktion INT-SAVE-CALL (FUNC, ARG1, ARG2 ...) definiert, die aufgerufen werden muß, um einen Rückruf durchzuführen. Die Aufgabe von SAVE-CALL, ist es, die Daten auf dem Stapselspeicher (Stack) einschließlich des Rücksprungvektors zu der Funktion, die SAVE-CALL aufgerufen hat, aber ausgenommen der Werte von FUNC, ARG1, ARG2 ... von Lese- und Schreibzugriffen innerhalb der Funktion FUNC zu schützen.

Grundsätzlich gibt es drei Lösungen für SAVE-CALL:

1. SAVE-CALL öffnet ein neues Stacksegment; das Kartenbetriebssystem verwaltet den Stackzugriff;
2. SAVE-CALL beschränkt den Schreib- und Lesezugriff auf das laufende Stack-Segment.

Dabei gibt es wieder zwei Möglichkeiten, nämlich:

2.1. Das Kartenbetriebssystem verwaltet den Stackzugriff.

2.2. Die Fernaufruf- und Fernrückkehrinstruktionen verwalten den Stackzugriff.

Fig. 2 zeigt die Ausführung eines sicheren Rücksprungs auf eine Funktion FUNC () in Vertrauensklasse 3 von einem Aufrufer in Vertrauensklasse 2. FUNC und seine Parameter werden als Parameter zu einer Betriebssystemfunktion SAVE-CALL übergeben. SAVE-CALL übergibt FUNC und seine Argumente weiter zu einer Betriebssystemfunktion ACTUAL SAVE CALL in Vertrauensklasse 3. Dieser Rücksprung ist nur zulässig, da sich SAVE-

CALL in Vertrauensklasse 1 befindet. ACTUAL SAVE CALL besitzt bereits einen geschützten Stack, wenn es FUNC aufruft. Nachdem FUNC zu ACTUAL SAVE CALL zurückgekehrt ist, wird RETURN SAVE CALL auf dem Kartenbetriebssystem mit Vertrauensklasse 1 aufgerufen. RETURN SAVE CALL gibt den Stack frei und ersetzt seinen Rücksprungvektor durch den Rücksprungvektor von SAVE CALL, der in einer Datei des Kartenbetriebssystems durch SAVE CALL selbst gespeichert worden ist. Sodann kehrt RETURN SAVE CALL zu der aufrufenden Funktion in der Bibliothek LIB zurück.

Erfindungsgemäß gibt es für die Realisierung der Funktion SAVE CALL zwei verschiedene Möglichkeiten:

15 1. SAVE CALL eröffnet einen neuen Stack oder ein neues Stacksegment.

Beim Aufruf von SAVE CALL übernimmt diese Funktion den Stack mit folgenden Inhalten:

20 Operanden, Argumente, Name der Funktion FUNC, Rücksprungadresse zu dem aufrufenden Programm in LIB.

Das Programm SAVE CALL führt dann folgende Aktionen aus: Es wird ein neues Datensegment DS eröffnet, die Argumente und der Name der Funktion FUNC werden in das neue Datensegment kopiert, die derzeitige Rücksprungadresse für den fernen Rücksprung SP mit 24 Bit Länge wird in eine Datei des Kartenbetriebssystems gespeichert, SP wird zu DS:LÄNGE gesetzt und 30 die Funktion ACTUAL SAVE CALL wird aufgerufen.

Die Funktion ACTUAL SAVE CALL übernimmt daher folgenden Stackinhalt bevor die Funktion FUNC aufgerufen wird:

35 Argumente, Name der Funktion FUNC, Rücksprungadresse zu dem Programm SAVE CALL.

Das Programm ACTUAL SAVE CALL führt nun folgende Aktionen durch:

Hole den Rücksprungvektor vom Stack, lade die Adresse von
5 FUNC in den Akku, rufe die Funktion FUNC indirekt auf.

Nachdem die Funktion FUNC ausgeführt worden ist, ist der Stack von ACTUAL SAVE CALL leer. Es wird sodann die Funktion RETURN SAVE CALL aufgerufen. Diese lädt das Register für die

10 Fernrücksprungadresse SP mit den ursprünglichen Werten, die in einer Kartenbetriebssystemdatei zwischengespeichert worden sind und löscht den zwischenzeitlich geschaffenen Stack oder das zwischenzeitlich geschaffene Stacksegment. Die Funktion RETURN SAVE CALL übergibt sodann den Stack wieder in der an-
15 fänglichen Belegung mit den Operanden, Argumenten, dem Namen der Funktion FUNC und dem Rücksprungvektor an das aufrufende Programm in LIB. Diese Rücksprungadresse wird nun in den Ak-
kumulator geladen und damit springt der Programmablauf in das aufrufende Programm zurück. Diese Vorgehensweise hat den Vor-
20 teil, daß sie bei allen denkbaren Strukturen des Stacks ange-
wendet werden kann. Es müssen jedoch die Argumente von FUNC kopiert werden und es muß eine Kartenbetriebssystemdatei zur Zwischen Speicherung der Rücksprungadresse angelegt werden.

25 2. Eine weitere erfindungsgemäße Lösung für sichere Rück-
sprungaufrufe führt eine Schreib/Lesebarriere auf dem Stack ein, die den Rücksprungvektor zu dem aufrufenden Programm vor Veränderungen schützt und ebenso den gesamten Stackinhalt des aufrufenden Programms von Schreib- und Lesezugriffen schützt.
30 Das kann durch eine geeignete Verminderung der Länge des Stacksegments oder durch ein zusätzliches Register in der Speicherverwaltung (MMU) erreicht werden. Ein Problem, welches dabei gelöst werden muß, ist, daß die Argumente einer Funktion vor dem Rücksprungvektor liegen, wenn das konven-
35 tionale C-Stack-Layout benutzt wurde. Eine Lösung dazu ergibt sich durch eine Neuordnung des C Stacks in einer solchen Wei-
se, daß Platz für den Rücksprungvektor reserviert werden muß,

10

bevor die Argumente auf den Stack gelegt werden. Dies führt zu einem zusätzlichen Programmieraufwand für einen Funktionsaufruf im Vergleich mit dem üblichen Stack Layout.

5 Im einzelnen erfolgt dieser weitere erfindungsgemäße Verfahrensablauf wie folgt:

Der Stack wird an SAVE CALL übergeben. SAVE CALL setzt eine Lese- und Schreibsperrre zwischen den Rücksprungvektor und die 10 Parameter von SAVE CALL. Der Stack hat dann folgenden Aufbau:

Operanden, Rückkehrvektor zu dem aufrufenden Programm in LIB, Rücksprung- Lese- und Schreibsperrre, Argumente, Name der Funktion FUNC.

15 Sodann wird das Programm ACTUAL SAVE CALL aufgerufen. Vom Programm ACTUAL SAVE CALL aus gesehen, besteht der Stackinhalt damit nur noch aus den Argumenten und dem Namen der Funktion FUNC, bevor die Funktion FUNC aufgerufen wird, da 20 die Funktion ACTUAL SAVE CALL nicht über die Lese- und Schreibsperrre hinauslesen kann.

Sodann wird zur Ausführung der Funktion FUNC der Rücksprungvektor vom Stack geholt, die Adresse der Funktion FUNC in den Akku geladen und die Funktion FUNC indirekt aufgerufen.

25 Bei Rückkehr aus der Funktion FUNC ist der Stack für die Funktion ACTUAL SAVE CALL leer. Die Funktion ACTUAL SAVE CALL ruft dann die Funktion RETURN SAVE CALL auf. Die Funktion 30 RETURN SAVE CALL löscht oder entfernt die Lese- und Schreibsperrre für den Stack, so daß der Stack für die Funktion RETURN SAVE CALL wieder folgenden Inhalt hat:

Operanden, Rücksprungvektor zu dem aufrufenden Programm in 35 LIB, Rücksprungvektor zu ACTUAL SAVE CALL. Von dem Programm RETURN SAVE CALL wird dann der Rücksprungvektor zu dem Programm ACTUAL SAVE CALL vom Stack geholt und die Stackrahmen-

zeiger werden zurückgesetzt. Das Programm ACTUAL SAVE CALL übergibt die Kontrolle dann an das aufrufende Programm in LIB.

5 Hierbei ist zu beachten, daß diese Vorgehensweise nur mit einer besonderen Struktur des Stacks möglich ist. Diese Vorgehensweise hat jedoch den Vorteil, daß die Argumente von FUNC nicht kopiert werden müssen, und keine zusätzlichen Dateien vom Betriebssystem angelegt werden müssen.

10 Weiter ist erfindungsgemäß noch die Lösung denkbar, daß eine bestimmte Anzahl von Argumenten in Register übergeben werden. Ein sicherer Rücksprungaufruf würde dann lediglich diese Anzahl von Argumenten als Maximum erlauben. Der Rücksprungvektor eines sicheren Rücksprungaufrufs besteht dann aus dem Rücksprungvektor eines normalen Funktionsaufrufs und zusätzlich aus der Länge des alten Stacksegments.

20 Weiterhin wäre es erfindungsgemäß auch möglich, den Rücksprungvektor auf einem separaten Stack zwischenzuspeichern. Dann kann die Lese- und Schreibsperrre einfach installiert werden, bevor das erste Funktionsargument auf den normalen Stack abgelegt wird.

25 Weiterhin ist noch eine erfindungsgemäße Lösung denkbar, bei der die gleiche Stackstruktur wie bei der vorstehenden Lösung 2 angewendet wird. Im Gegensatz zu den ersten beiden Lösungen schützt jedoch nicht das Kartenbetriebssystem, sondern das aufrufende Programm selbst den Stack des aufrufenden Programms durch einen besonderen Befehl SEC CALL gegen Lese- und Schreibzugriff. Bei der Ausführung des Rücksprungbefehls muß dann geprüft werden, ob der Rücksprungvektor hinter der Lese- und Schreibbarriere liegt. Wenn dies der Fall ist, kann die alte Lese- und Schreibbarriere, die auf dem Stack hinter der aktuellen Barriere gespeichert worden ist, wieder hergestellt werden und der übliche Rücksprung ausgeführt werden. Andernfalls wird nur der übliche Rücksprung ausgeführt. Im Hinblick

auf die Struktur des Stacks entspricht diese Lösung der vorher beschriebenen Lösung mit der besonderen Stackstruktur.

Die Fig. 1 zeigt das einfachste Grundprinzip aller dieser erfundungsgemäßen Lösungen:

Der Stackzugriff muß bei einem Aufruf einer unsicheren Funktion auf deren Stackbereich durch Hardware beschränkt werden. Man erreicht dies durch Speichern des Stackframepointers der aufrufenden Funktion. Dabei ist ein Schutzmechanismus zu implementieren, so daß die aufgerufene Funktion den Wert des gespeicherten Stackframepointers nicht verändern kann. Außerdem ist beim Schreiben auf den Stack sicherzustellen, daß der Stackpointer nicht den gültigen Stackbereich der aktuellen Funktion überschreitet.

Der Schutzmechanismus kann automatisch aktiviert werden oder von der aufrufenden Funktion direkt angestoßen werden.

Beim RETURN aus der unsicheren Funktion wird dabei durch eine Hardwareimplementation der ursprüngliche Zustand auf dem Stack wieder hergestellt.

Entsprechend zeigt die linke Darstellung in Fig. 1 den Zustand vor dem Funktionsaufruf. Der Stackpointer SP zeigt auf die oberste belegte Speicherzelle im Stack. Unterhalb davon ist der Stack belegt, es darf aber auf den Stack zugegriffen werden. Der Stackframepointer SFP ist nicht belegt oder enthält einen Wert für eine Sperre aus einem früheren Aufruf.

Die rechte Darstellung der Fig. 1 zeigt den Zustand nach dem Funktionsaufruf. Der Stackpointer zeigt nun auf eine Speicherzelle weiter oben, die als letzte belegt ist. Nach dieser oder diesen belegten Speicherzellen liegt als letztes die aufgerufene Funktion FN und ihre Argumente (ARG). Der Framepointer zeigt auf ein Feld, in dem der alte Wert des Stackframepointers zwischengespeichert ist (im Falle von mehrfa-

chen geschützten Funktionsaufrufen), oder das leer ist. Die Speicherzelle, auf die der Stackframepointer zeigt, ist automatisch für den Zugriff gesperrt, da Zugriffe nur zulässig sind, wenn $SP < SFP$. Damit ist diese Speicherzelle und alle darunter folgenden auch gegen Manipulationen gesichert.

Beim Funktionsaufruf wird zunächst auf dem Stack ein Speicherbereich für zu schützende Funktionsdaten reserviert (Rücksprungadresse, etc). Anschließend werden die Funktionsargumente auf den Stack gelegt. Schließlich wird beim Funktionsaufruf der im geschützten Speicherbereich liegende SFP (mit dem Stackframepointer der aufrufenden Funktion der aktuellen Funktion) auf den zuvor reservierten Bereich des Stacks gelegt und der Stackframepointer der aktuellen Funktion in den geschützten Bereich geschrieben (SFP). Dies erfolgt entweder durch Betriebssystemaufruf oder durch einen Hardwaremechanismus.

Bei Stackmanipulationen wird der Stackpointer stets mit dem abgespeicherten Wert SFP im geschützten Bereich verglichen, um zu verhindern, daß der Stackbereich der aufrufenden Funktion manipulierbar wird.

Erfindungsgemäß handelt es sich also um eine Hardware unterstützte Lösung zur Absicherung des Stacks der aufrufenden Funktion gegen Überschreiben. Der Rücksprung in die sichere Funktion und optional auch der Aufruf der unsicheren Funktion kann dabei ohne Interaktion mit dem Betriebssystem erfolgen. Dies bedeutet einen Geschwindigkeitsvorteil bei unsicheren Funktionsaufrufen.

Die Alternative wäre, den Funktionsaufruf nicht direkt auszuführen, sondern den Funktionspointer und die Argumente der Funktion an das Betriebssystem zu übergeben, das den bisherigen Stack absichert und anschließend die Funktion aufrufe. Dies ist jedoch sehr kompliziert und rechenzeitaufwendig.

Durch die vorliegende Erfindung wird also die Implementation eines sicheren Callbacks in C und für die Java Virtual Machine auf eine Chipkarte deutlich erleichtert. Unter anderem kann der Umweg über das Betriebssystem, der meist ein großes
5 Handicap darstellt, erspart werden.

Patentansprüche

1. Verfahren zur Verhinderung von Stackmanipulationsangriffen bei Funktionsaufrufen, d a d u r c h g e k e n n z e i c h -
5 n e t, daß der Stackzugriff bei einem Aufruf einer unsicheren Funktion durch Hardware auf den Stackbereich dieser unsicher-
ren Funktion beschränkt wird.

10 2. Verfahren nach Anspruch 1, d a d u r c h g e k e n n -
z e i c h n e t, daß zur Beschränkung des Stackzugriffs vor
dem Aufruf der unsicheren Funktion eine Referenz auf den
Stackframe der aufrufenden Funktion gespeichert wird.

15 3. Verfahren nach Anspruch 2, d a d u r c h g e k e n n -
z e i c h n e t, daß ein Mechanismus vorgesehen ist, durch
den verhindert wird, daß die aufgerufene Funktion auf den Wert der Referenz auf den Stackframe und alle davor auf dem Stack liegenden Daten zugreifen kann.

20 4. Verfahren nach einem der Ansprüche 1, 2 oder 3, d a-
d u r c h g e k e n n z e i c h n e t, daß durch einen Schutzmechanismus sichergestellt wird, daß der Stackpointer nicht den gültigen Stackbereich der aufgerufenen Funktion überschreitet.

25 5. Verfahren nach einem der Ansprüche 1 bis 4, d a d u r c h g e k e n n z e i c h n e t, daß bei dem Rücksprung aus der unsicheren Funktion der ursprüngliche Zustand auf dem Stack wieder hergestellt wird.

30 6. Verfahren nach einem der Ansprüche 1 bis 5, d a d u r c h g e k e n n z e i c h n e t, daß beim Funktionsaufruf zu-
nächst auf dem Stack ein Speicherbereich für zu schützende Funktionsdaten reserviert und optional dahinter die Funkti-
35 onsargumente auf den Stack gelegt werden können und die im geschützten Bereich liegende Referenz auf den Stackframe der aufrufenden Funktion auf den zuvor reservierten Bereich des

Stacks gelegt und die Referenz auf den Stackframe der aufgerufenen Funktion in den geschützten Bereich geschrieben wird.

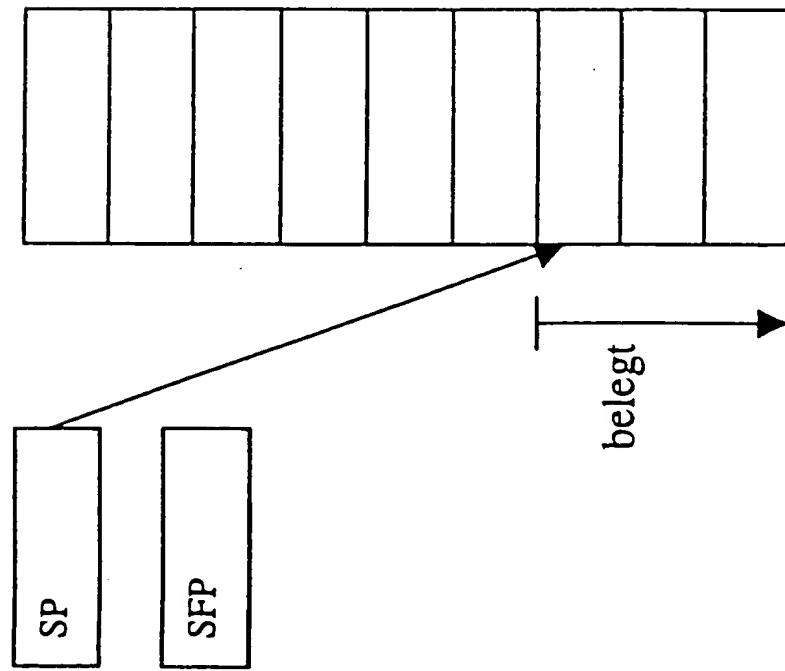
Zusammenfassung

Verfahren zur Verbindung von Stackmanipulationsangriffen bei Funktionsaufrufen

Hardwareunterstütztes Verfahren zur Verhinderung von Stackmanipulationsangriffen bei Funktionsaufrufen, bei dem der Stackzugriff bei einem Aufruf einer unsicheren Funktion durch Hardware auf den Stackbereich dieser Funktion beschränkt wird.

Fig. 1

vor dem Funktionsaufruf:



nach dem Funktionsaufruf:

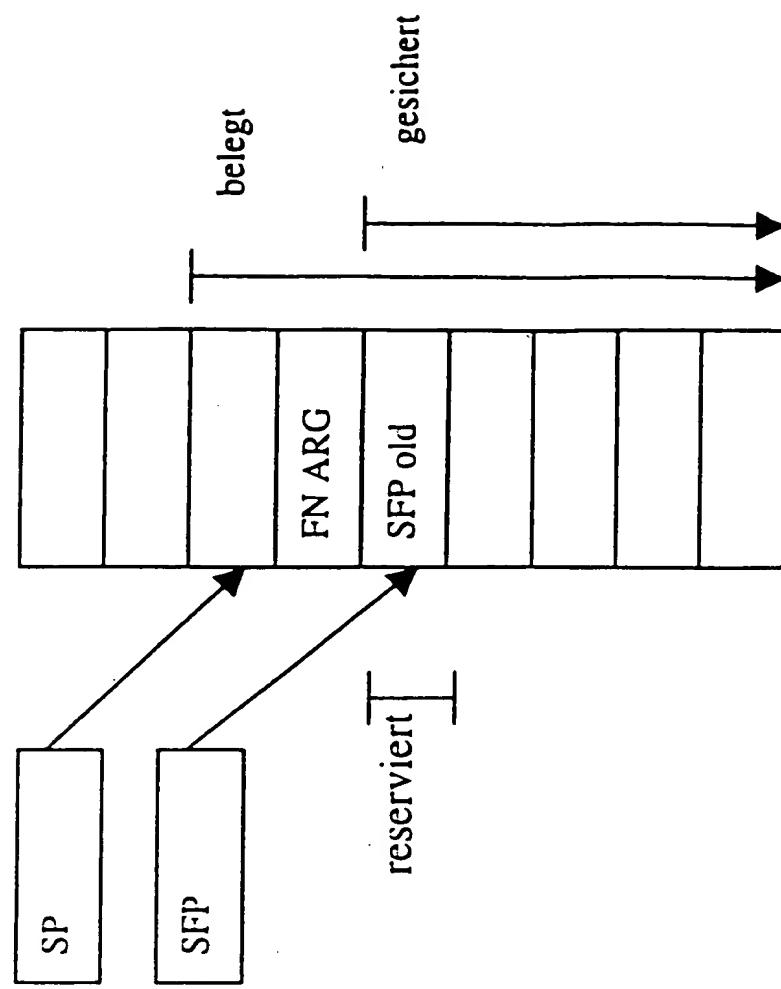


Fig. 1

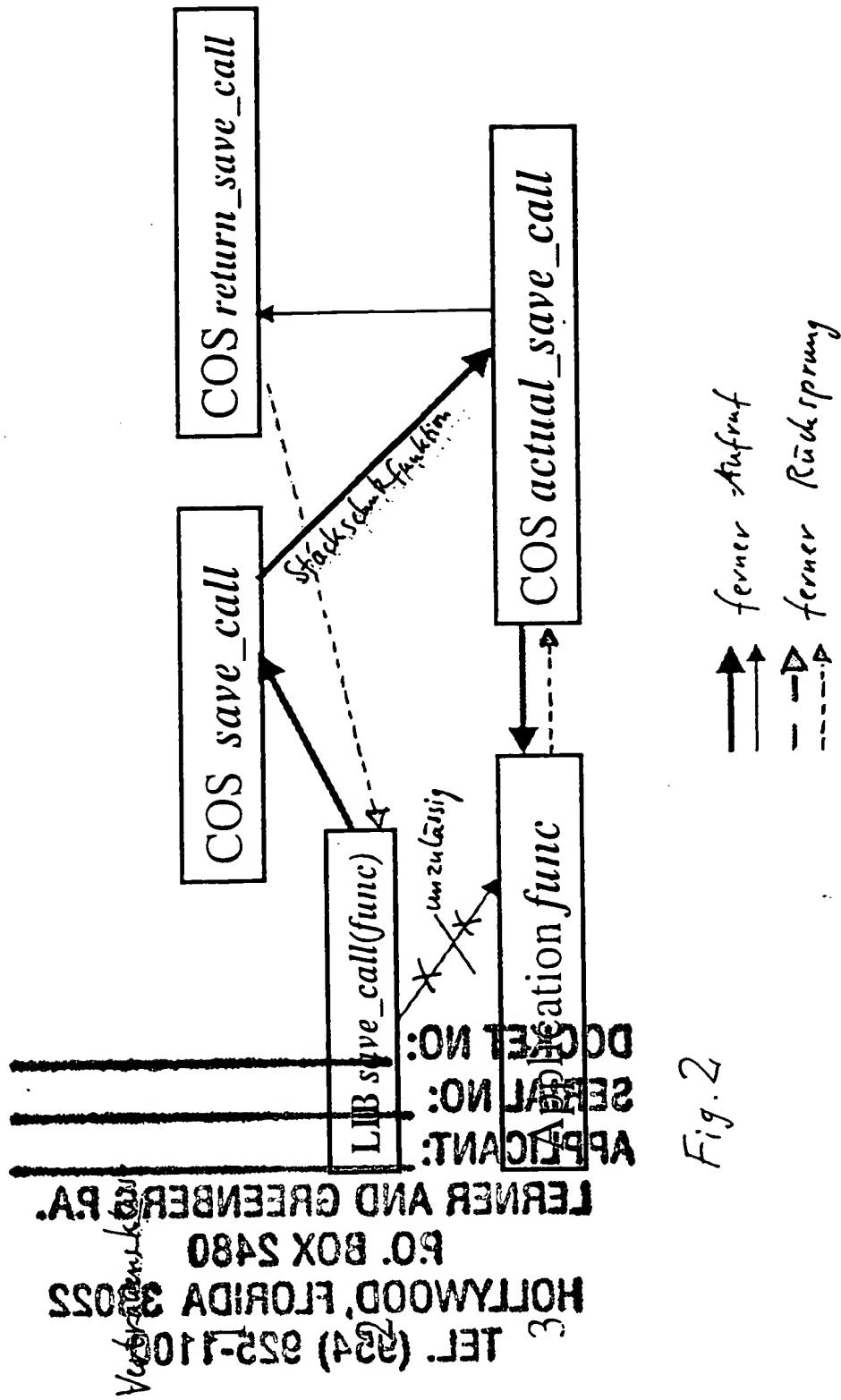


Fig. 2